

# Конференция клиентских интернет-технологий Client Side'2007

---

**С добрым утром!**

Мы сидим в Кофемании, за неделю до начала конференции и мучительно думаем о том, что написать в приветствии. Это третья конференция российского интернет-сообщества, поворотный момент. Драйв в организации меняется на спокойную работу профессионалов, времена золотоискателей потихоньку проходят.

Конференция клиентских технологий обо всем, что видит пользователь на экране своего браузера. По сути – это цель нашей работы, вторая сторона качества. Этот вопрос мы тоже должны проработать, если, конечно, мы хотим быть лучшими и продавать еще что-нибудь, кроме нефти.

21 век, экономика информации плавно превращается в экономику внимания. Вопрос лидерства открыт – кто будет управлять 21 веком, кто войдет в историю? Какие нации, чьи открытия оставят наибольший след? Кто будет устанавливать правила игры?

Россия должна быть с теми, кто думает. У нас всегда будет энергия, теперь нам нужно, чтобы наши ученые работали здесь. Нужно формировать элиту, которая думает об эволюции, техническом прогрессе, вот наша задача. Мы должны заново отстроить научные городки, студенческое братство. Нам нужно отнять у Америки идею страны больших возможностей. Потому что если и быть центру технологий, центру открытий, центру ноу-хау, центру разума в XXI веке, то быть ему именно в России.

У Вас в руках краткое содержание докладов Client Side'2007. В этот раз двадцать ведущих специалистов Рунета отбирали для Вас лучшие выступления из всех присланных материалов. Конкуренция – залог качества ;-)

Давайте хорошо поработаем эти два дня и вернемся к нашим пользователям с новыми идеями и свежими знаниями. И сделаем наш мир еще чуточку лучше и удобнее!

Удачи и больших Вам целей!

*Олег Бунин, Павел Рогожин, Дмитрий Филев,  
команда Client Side'2007,  
<http://www.client2007.ru/>*

# Содержание

Конференция клиентских интернет-технологий Client Side'2007 .....	1
Содержание .....	3
Сборник тезисов .....	6
<b>Секция «AJAX, JavaScript, JSON» .....</b>	<b>6</b>
Отладка JavaScript/Ajax и профилирование.....	6
Оффлайн веб-приложения: от Google Gears до HTML5 .....	6
Произвольные события — клей сложных веб-приложений .....	8
Разработка, оптимизация и тестирование тонких клиентов .....	8
Практическое использование AJAX .....	10
Мастер-класс «Модульность в JavaScript» .....	11
Мастер-класс «Векторная графика в Вебе (SVG, VML, Canvas)» .....	11
<b>Секция «Производительность» .....</b>	<b>12</b>
Управление скоростью реакции сайта .....	12
Оптимизация HTML, CSS, js на часто показываемых страницах (на примере морды и результатов поиска Яндекса) .....	13
Особенности верстки крупных проектов .....	17
<b>Секция «Мультимедиа: Flash, AIR, Flex, SilverLight, GWT» .....</b>	<b>18</b>
Сравнение современных технологий для создания насыщенных интернет-приложений (RIA) .....	18
Adobe Integrated Runtime (Adobe AIR): платформа для создания десктопных RIA.....	20
Работа с данными в формате XML в среде Adobe Flash. Создание standalone приложений с использованием Adobe Flash и Northcode SWF .....	21

Технология Google Web Toolkit .....	21
Анимационные эффекты средствами браузера .....	23
Мастер-класс «Геометрия во Flash: кривые Безье второго порядка» ...	24
<b>Секция «Веб-стандарты, верстка, HTML 5.0» .....</b>	<b>25</b>
Семантическая вёрстка .....	25
Веб-стандарты в ASP.NET .....	25
Как сделать сайт доступным? .....	26
Верстка независимыми блоками .....	26
Тонкий CSS для Internet Explorer .....	27
HTML 5.0.....	27
Мастер-класс «Научим верстать быстро, качественно, дорого...» .....	28
<b>Секция «Веб-дизайн» .....</b>	<b>29</b>
Нюансы веб-типографики .....	29
Графический дизайн для веб-среды .....	30
Управление проектами. Информационный дизайн .....	30
Как выжить дизайнеру в веб 2.0 стартапе .....	31
Мастер-класс «Интерфейс на экране, на бумаге и в жизни» .....	32
<b>Секция «Дизайн интерфейсов, юзабилити, accessibility» .....</b>	<b>33</b>
Качество человеко-компьютерного взаимодействия: подход в международных стандартах .....	33
Состояние потока (FLOW) как цель проектирования взаимодействия ...	33
Расширенное понимание доступности, ее основные составляющие и методы обеспечения .....	35

Usability-проектирование. Dark Side of the Source .....	38
Адаптация продуктов Microsoft для людей с ограниченными возможностями (accessibility) .....	42
Особенности проектирования интерфейсов для начинающих пользователей .....	43
Мастер-классы «Юзабилити-тестирование? Легко! (или как в домашних условиях протестировать программный продукт)» .....	44
Мастер-класс «Юзабилити аспекты проектирования пользовательских интерфейсов в среде Microsoft Expression Blend» .....	44
<b>Секция «Шаблонизаторы» .....</b>	<b>46</b>
Почему XSLT? .....	46
Эффективное использование XSLT .....	46
Как проще и эффективнее строить работу с унаследованным от других разработчиков xsl-кодом .....	47
Мастер-класс «Применение XSLT на стороне клиента» .....	48
<b>Секция «Мобильные технологии (WAP, PDA-проекты)» .....</b>	<b>49</b>
Юзабилити программ для мобильных устройств .....	49
Мобильные магазины и платежи .....	50
<b>Секция «Качество, тестирование» .....</b>	<b>52</b>
Автоматизация тестирования веб-интерфейсов с использованием Selenium .....	52
Автоматизация тестирования сложных Flash-интерфейсов .....	52
Что такое XSS и как их искать .....	53
Автоматизированное тестирование смешанных web- и win-интерфейсов .....	54

---

# Сборник тезисов

---

## Секция «AJAX, JavaScript, JSON»

### Отладка JavaScript/Ajax и профилирование

*Александр Шуркаев*

#### **История вопроса.**

Языковые конструкции для отладки: alert(), try/catch, onerror.

Инструментарий для отладки и профилирования:

1. кросс-браузерные средства (Firebug Lite, JsLex)
2. браузерные средства:
  - a. Mozilla/Firefox (Firebug, YSlow, Venkman);
  - b. Internet Explorer (Microsoft Script Debugger, Companion.JS, DebugBar);
  - c. Opera (JavaScript Console);
  - d. Safari (JavaScript Console).

Development- и production-логирование (Jash и прочее).

### Оффлайн веб-приложения: от Google Gears до HTML5

*Алексей Захлестин*

#### **История вопроса**

Статичные ресурсы. Архитектура «запрос-ответ». Второе пришествие тонких клиентов (AJAX). Параллели с традиционным программным обеспечением.

#### **Цель**

Проектирование веб-приложений сегодня движется в сторону максимального переноса логики с сервера на клиент. Серверу остаётся роль агента ав-

торизации и синхронизации. Возникает проблема хранения локальных данных (ограничения cookies). Свои решения предлагают закрытые технологии: Flash, Silverlight. Где же выход?

### **Поворотный момент**

30 мая 2007 года Google анонсировала свой проект с открытым исходным кодом — Google Gears™, выразив намерение выработать открытые стандарты для решения задач такого рода. В тот же день идею стандартизации поддержала компания Apple, начав обсуждение в рамках WHAT-WG. Было принято решение ввести стандартизацию веб-приложений, работающих без подключения к сети в рамках спецификации Web Applications 1.0, также известной как HTML 5.

### **Части мозаики**

#### **Кэширование приложений**

В настоящий момент, браузеры используют так называемую «наивную» модель кэширования данных, при которой в памяти сохраняются только те данные, которые уже были явно или неявно (с помощью разнообразных технологий предварительной загрузки) отображены. Такая модель не подходит для оффлайн-приложений — им гораздо удобнее иметь возможность указать список необходимых ресурсов и добавлять часть данных по ходу работы.

#### **Локальные данные**

Для полноценной работы в условиях отсутствия подключения приложению необходима возможность локального хранения большого количества структурированных данных. Спецификация Web Applications 1.0 определяет два типа таких хранилищ:

1. ассоциативные массивы;
2. базы данных.

#### **Undo/Redo**

Спецификация определяет стандартный интерфейс для работы с состояниями приложения.

#### **Отложенная загрузка файлов на сервер**

Приложениям, работающим в условиях отсутствия подключения, часто бывает необходимо использовать файлы, предоставленные пользователем (например, текстовый редактор или клиент электронной почты)

## Переходим к практике

В этой секции будет приведен пошаговый пример создания веб-приложения, которое может работать как при наличии подключения, так и без него. Скорее всего, это будет несложный текстовый редактор.

## Платформы

Бета-версия расширения Google Gears™ для Firefox доступна уже сейчас (будет и для IE?). Вероятно, будет поддержка и в Opera 10, Safari 4, Mozilla 2/Firefox 4. Массовое внедрение технологии предполагается к середине 2008 года?

## Произвольные события — клей сложных веб-приложений

*Андрей Сумин*

Современное веб-приложение — не просто сумма сложных компонентов на странице, это еще и их взаимосвязь. Современные приложения делают ее еще более сложной за счет динамических интерфейсов (AJAX).

В большинстве веб-приложений есть повторяющиеся элементы, которые разработчик хочет сделать один раз и использовать во многих местах без модификации программного кода.

Невозможно предугадать все варианты внешних связей единожды написанного элемента, но можно определить список его состояний и дать возможность сообщать окружению о текущем его состоянии или изменении.

Произвольные события позволяют компоненту встраиваться в интерфейс любой сложности. Проксирующие объекты (объекты обертки), использующие эти события, могут серьезным образом изменить поведение компонента, не меняя его программный код.

Большинство современных библиотек уже позволяют создавать произвольные события, что говорит о полезности такого подхода к созданию сложных интерфейсов веб-приложений.

## Разработка, оптимизация и тестирование тонких клиентов

*Петр Макаров,  
Иван Кузнецов*

## Введение

Кто мы и почему об этом рассказываем. Также о том, почему мы не можем рассказать вам всего, что хотим.

## Почему тонкий клиент?

По сравнению с сайтами:

1. более высокая скорость работы и удобство интерфейса, делающее его более дружелюбным пользователю:
  - каждое действие отправляется на сервер;
  - отсутствуют перезагрузки страниц (меньшее количество запросов к серверу; сокращение времени на рендеринг);
  - отсутствие необработанных серверных ошибок;
  - независимость от серверной части.

По сравнению с приложениями:

1. реальная кроссплатформенность;
2. возможность быстрых обновлений и багфиксов (отсутствие развертывания приложения на стороне клиента);
3. простота разработки.

## Практические задачи

1. Объем загружаемых данных при первой загрузке менее 100кб:
  - динамическая загрузка JS. При начала работы достаточно минимального количества данных, остальные скрипты загружаются по необходимости (например, обработчики ответов сервера);
  - высокая скорость ответа на действия пользователя;
  - динамическая загрузка шаблонов.
2. Низкое время отклика интерфейса на действия пользователя:
  - JSON в качестве транспорта (меньшее количество трафика по сравнению с XML; высокая скорость обработки средствами JavaScript; развивающиеся решения с открытым исходным кодом, упрощающие работу с этим форматом).
3. Простота внесения изменений в условиях меняющихся требований:
  - JS-шаблонизаторы как кроссплатформенное средство шаблонизации:
    - a. не требуют наличия библиотек на стороне клиента, как, например, XSLT; как следствие – большая кроссбраузерность;
    - b. в отличие от построения дерева и работы с DOM, позволяют быстрее и нагляднее вносить изменения, поддерживать меньше кода;
    - c. в отличие от передачи HTML со стороны сервера, не зависят от серверной реализации и отделяют клиентскую часть от серверной;
    - d. пример простого шаблонизатора: replace в цикле и кэш.

- Тестирование UI:
    - a. юнит-тестирование JavaScript и протокола;
    - b. приемочные кроссплатформенные тесты Selenium, кратко о Selenium.
  - Ну и не забывайте о подготовке требований и итеративной разработке ;-)
4. Развертывание
1. Компрессия кода клиента, сборка (вы знаете что «;» перед закрывающей фигурной скобкой в CSS не так уж обязательна, а длинные имена директорий и названий файлов — тоже трафик).

### Архитектура одного проекта

1. Диспетчер, отвечающий за централизованную передачу данных.
2. Контроллер для централизованного хранения шаблонов запросов.
3. «Ленивая» загрузка шаблонов.
4. «Ленивая» загрузка данных.
5. «Ленивая» загрузка кода.
6. Шаблонизация.

### Развитие проекта или разговор о том, как сделать проект еще лучше

1. Постоянно увеличивается покрытие тестами.
2. Самописные библиотеки заменяются открытым исходным кодом.
3. Постоянный рефакторинг.

## Практическое использование AJAX

*Докладчик уточняется*

Как и насколько применяется AJAX в Битрикс:

- a) повысило:
  1. привлекательность редакторского интерфейса;
  2. удобство работы редакторов;
  3. производительность труда редакторов (выставляют, изменяют и перемещают больше контента, чем раньше?);
  4. удовлетворенность редакторов от работы.
- b) понизило:
  1. трудозатраты на выполнение редакторских задач в интерфейсе (где требовались два редактора, там теперь хватает одного);
  2. количество ошибок, совершаемых операторами-редакторами;

3. вероятность потерь данных (за счёт частого сохранения?);
4. расходы клиента на оплату труда редакторов.

## Мастер-класс «Модульность в JavaScript»

*Андрей Сумин*

Веб-приложения перестали быть HTML-страницами, они насыщены большим количеством JavaScript-кода. На этом мастер-классе я покажу, как решал следующие проблемы:

1. автоматизация подключения только нужных JavaScript-файлов на странице;
2. использование единого кода на разных проектах;
3. исключение необходимости написания JavaScript-кода серверными программистами.

Для автоматизации подключения только нужных JavaScript-файлов я написал функционал, позволяющий указывать межфайловые зависимости. Этот функционал автоматически загружает нужные файлы, если какой-то JavaScript-код их требует.

Все файлы разбиты по namespace, у каждого namespace есть свой файл описания. Это позволяет собирать функционал из различных мест под отдельный проект.

Создана система компонентов, что позволяет серверным программистам объявлять узлы DOM-дерева компонентами и не заботиться что об инициализации, что о загрузке JavaScript-файлов.

## Мастер-класс «Векторная графика в Вебе (SVG, VML, Canvas)»

*Леонид Хачатуров*

1. SVG, VML, Canvas. Что это?
2. Возможности (базовые графические примитивы и их свойства, преобразования, обработка событий).
3. Как достигается кроссбраузерность (фреймворки)?
4. Области применения (эффекты, карты, графики и sparklines, динамическая графика и т.д.).
5. Эффекты.
6. Графики: timeline, timeplot.
7. Фреймворки общего назначения (dojo.gfx, Prototype Graphic Framework).
8. Проблемы и сложности (производительность, различия в реализациях SVG и VML).

## Секция «Производительность»

### Управление скоростью реакции сайта

*Артёмий Трегубенко,  
Николай Мацевский*

Составляющие скорости: время загрузки/инициализации новой страницы, время реакции элементов на странице.

Загрузка/инициализация: сетевые задержки, отрисовка HTML/CSS, инициализация скриптов.

Коротко о сетевых задержках: что уменьшать (dns, число запросов, размер запроса, размер ответа)?

Число запросов: необходимые запросы (кэширование, HTML, CSS, JavaScript, изображения).

Число запросов: склеивание (весь CSS в одном файле, весь JavaScript в одном файле, мелкие иконки в одном файле).

Число запросов: экстремальные варианты (CSS и JavaScript в одном файле, inline CSS/JavaScript (Yahoo), упрощение и уменьшение страниц (Google: только 10 результатов ради скорости)).

Размер запроса — уменьшение размера и количества cookies.

Размер ответа: gzip и minify.

Отрисовка HTML/CSS; веб-приложения: моментальный индикатор загрузки, максимум готового HTML; простые страницы быстрее; expressions; многое зависит от браузера, оптимизируют производители; CSS; JavaScript подключается после основного контента.

Проблемы подключения js: потенциальный `document.write`, `defer=»defer»`, парсинг на каждой загрузке.

Подключение js после контента: загрузка js не завершилась/прервана, progressive enhancement.

Инициализация скриптов: минимум действий.

Вредные советы: создавайте DOM-объекты про запас, даже если они не будут использованы; существенно меняйте страницу, много reflow; обрабатывайте побольше элементов.

Подключение обработчиков: обработка многих элементов, скорость поиска элементов, `behavior.js`: красиво, но медленно.

Глобальные обработчики: `event bubbling`, нет необходимости искать, динамически созданные элементы, нагрузка на процессор смещается с инициализации страницы к действиям пользователя, минимум действий.

Пример загрузочных скриптов.

Потенциал роста – `behavior` на глобальных обработчиках.

Время реакции элементов на странице: `click` и `hover`.

`Click vs. mousedown`: моментальная реакция, возможность отмены решения, индикатор загрузки, время для отрисовки.

Регресс: `addEventListener` vs. `onclick`.

Случайный `mouseover/mouseout`: точность действий человека невелика, реакция на случайный `hover` отвлекает, выпадающие меню и случайный `mouseout`.

Увеличение времени реакции `mouseover/mouseout` – срабатывание только при необходимости.

Пример реализации «медленного `hover`»: `Function.getDelayedHandlers()` и `Element.delayedHover()`.

Утилиты: `Firebug` / `MSVS`, `YSlow`.

## Оптимизация HTML, CSS, js на часто показываемых страницах (на примере морды и результатов поиска Яндекса)

*Александр Мусеев*

Страница с результатами поиска Яндекса за последние 12 месяцев показывалась почти 7 миллиардов раз. Очевидно, что такая посещаемая страница должна быть очень легкой, ведь пользователям комфортнее работать с быстро загружающимся ресурсом, а нам приятно экономить пользовательский трафик.

Пути оптимизации:

1. оптимизация разметки;
2. оптимизация стилей;

3. оптимизация скриптов;
4. другие способы оптимизации.

### Оптимизация разметки

Использование HTML вместо XHTML.

XHTML, являясь подмножеством XML, имеет более строгие требования к синтаксису; HTML допускает более свободную запись, этим можно воспользоваться.

1. Элементы, не требующие закрывающего тэга нужно писать без «/» - area, base, br, col, hr, img, input, link, meta, param (все элементы, у которых указано F в столбце End Tag в таблице <http://www.w3.org/TR/HTML4/index/elements.HTML>).
2. Не указывать элементы body, head, HTML, tbody (все элементы, у которых указано O в столбце Start Tag в таблице <http://www.w3.org/TR/HTML4/index/elements.HTML>).
3. Не указывать закрывающие тэги colgroup, dd, dt, li, option, p, td, tfoot, th, thead, tr (все элементы, у которых указано O в столбце End Tag в таблице <http://www.w3.org/TR/HTML4/index/elements.HTML>).
4. В некоторых случаях возможно писать значения атрибутов без кавычек: в таких случаях атрибуты могут содержать только английские буквы, цифры, дефис (ASCII код 45), точку (ASCII код 46), подчеркивание (ASCII код 95) и двоеточие (ASCII код 58). (См. <http://www.w3.org/TR/HTML4/intro/sgmltut.HTML#h-3.2.2>).
5. Не использовать пробел для разделения значений атрибутов, если значение взято в кавычки, т.е. вместо title=»На главную страницу» alt=»Логотип» использовать title=»На главную страницу»alt=»Логотип».

Удобнее всего писать в шаблоне нормальный HTML, а оптимизацию производить над шаблоном, который будет выкладываться в production. Ваша разметка по-прежнему будет оставаться валидной как в рамках HTML 4.01 Transitional, так и более жёсткого HTML 4.01 Strict.

Значения атрибутов по умолчанию.

Не использовать атрибуты, значения которых подразумевается по умолчанию: method=»get», target=»\_self», type=»text»

Заменять мнемонические сущности «&имя;» или «&#NNNN;» на их действительное представление.

Например, для документа в кодировке windows-1251, замена 109 сущностей «&nbsp;» на символ с кодом 160 уменьшает документ на 545байт. Для кодировки windows-1251 можно таким образом заменить

самые распространенные сущности: «&nbsp;», «&mdash;», «&laquo;», «&raquo;».

А, например, сущность «&bull;» мы не сможем заменить в документе с кодировкой windows-1215, так как её нет в кодовой таблице windows-1251, но если кодировка документа utf-8, мы сможем заменить любую сущность на ее представление.

Удаление из документа ненужных пробелов, переводов строк и комментариев.

Возможно организовать удаление после формирования страницы шаблонизатором или непосредственно в шаблоне перед выкладкой в production.

Использование специально оптимизированной разметки вместо универсальной.

Часто в крупных проектах используются универсальные шаблоны, которые используются на очень многих страницах. Как правило, это решения, в которых больше внимания уделено универсальности, нежели размеру кода.

### **Оптимизация стилей**

Очистка разметки от inline-стилей, перенос их в CSS-файл.

Придание именам классов коротких названий.

Во многих случаях можно обойтись более специфичным селектором в CSS, нежели давать имена классов большому количеству элементов.

Использовать один CSS-файл на странице/проекте.

Если это не подходит, то собирать автоматически несколько CSS-файлов в один. Один файл быстрее загружается, так как браузеру необходимо делать меньше http-запросов.

«Сжимать» CSS-файл.

При «сжатии» из файла удаляются бесполезные пробельные символы, переводы строк и комментарии. Мы для этих нужд используем утилиту, которая написана Виталием Харисовым. Пример «сжатого» файла: [http://CSS.yandex.net/CSS/\\_images.CSS](http://CSS.yandex.net/CSS/_images.CSS)

Из общедоступных утилит для сжатия CSS можно порекомендовать YUI Compressor. <http://www.julienlecomte.net/yuicompressor/>

**Оформлять CSS-правила наиболее лаконично.**

1. `color:#006600;` -> `color:#060;`
2. избавиться от `rgb()` в пользу `#hex`;
3. использовать, где это уместно, сокращенные формы записи: `background`, `border`, `font`, `margin`, `padding`.

**Оптимизация скриптов**

По возможности вынести все встроенные (`embedded`) и инлайновые (`inline`) стили в `js`-файл.

Давать переменным и функциям, по возможности, короткие, но понятные названия; в некоторых местах уместно писать `0` и `1` вместо `false` и `true`.

Использовать один `js`-файл на странице/проекте.

«Сжимать» `js`-файлы.

При «сжатии» из файла удаляются бесполезные пробельные символы, переводы строк и комментарии. Для этих нужд можно использовать утилиту `JSMIn` (<http://www.crockford.com/JavaScript/jsmin.HTML>), либо `YUI Compressor` (<http://www.julienlecomte.net/yuicompressor/>).

`YUI Compressor` может не только вырезать лишние пробелы и переводы строк, но и делать обфускацию переменных, сокращая их.

**Другие способы оптимизации**

Оптимизация графики в таких программах как `Adobe Fireworks` (`Adobe Image Ready`).

Если это позволяет разметка, объединять небольшие картинки в один файл и в разметке отрисовывать их как `CSS background` с разными значениями `background-position`

Ресурсы, которым не нужны куки, вынести в другой домен.

Очевидно, что картинкам (как и `CSS` и `js`-файлам) куки не нужны, поэтому, положив их в другой домен, мы дешевым способом экономим исходящий трафик пользователя. Посмотрите на `http get`-запрос картинки из домена `.ru` (ставятся куки) из домена `.net` (куки не ставятся).

Включить кэширование статики на сервере.

Включить «зипование» `HTML`-, `CSS`-, `js`-файлов на сервере.

## Особенности верстки крупных проектов

*Александр Тимофеев*

Различия принципов верстки «больших» проектов и «маленьких» сайтов:

- код страниц и файлов стилей должен быть хорошо структурирован и предусматривать возможность расширения;
- код должен быть не только понятным, но и компактным;
- нужно учитывать особенности работы в команде и придерживаться внутренних стандартов кодирования;
- глобальные изменения вносить сложно, а иногда и опасно;
- приходится ориентироваться на большее количество возможных браузеров.

Некоторые приемы верстки крупных проектов:

- формирование глобальной структуры страницы;
- правильное подключение стилей;
- использование общих файлов;
- оптимизация кода таблиц стилей: ручная оптимизация, программы и сервисы, оптимизация на стороне сервера.

## Секция «Мультимедиа: Flash, AIR, Flex, SilverLight, GWT»

### Сравнение современных технологий для создания насыщенных интернет-приложений (RIA)

*Константин Ковалев*

В настоящее время наблюдается устойчивая тенденция обогащения интернет-приложений различными интерактивными, визуальными и мультимедийными особенностями, которые ставят такие приложения в один ряд с традиционным программным обеспечением. Введенный фирмой Macromedia в марте 2002 года термин RIA (Rich Internet Applications — насыщенные интернет-приложения), стал устойчивым; сформировался соответствующий рынок, на котором появились довольно сильные игроки. Также меняются и способы доставки RIA клиенту.

Известный большинству интернет-пользователей Flash-плеер от Macromedia (затем бренд был куплен компанией Adobe), прочно завоевал свое место в предоставлении интерактивного контента в интернете, который обогащен анимацией и мультимедиа-возможностями. В настоящее время Flash-плеер установлен более чем на 98% компьютеров. В 2004-м году появился термин «Flash-платформа», который знаменовал собой выход за рамки традиционной среды разработки Flash и появление ряда RIA-технологий, из которых в докладе рассматриваются Adobe Flex и Adobe Integrated Runtime (Adobe AIR).

Adobe Flex – мощный фреймворк, компилятор и набор инструментов, которые позволяют создавать традиционные Flash-приложения с использованием декларативного языка разметки MXML.

Adobe AIR представляет собой технологию создания RIA в традиционном ПО, интегрирующую в себя Flash/Flex, HTML/DHTML/JavaScript/AJAX и PDF.

При создании операционной системы Windows Vista, Microsoft решила привнести в нее новый подход к созданию пользовательского интерфейса. Проект под кодовым названием Avalon в ноябре 2006 вышел как Windows Presentation Foundation (WPF) — часть .NET Framework 3.0, доступного не только на Windows Vista, но и на более ранних версиях Windows. WPF предлагает основанный на XML декларативный язык разметки XAML для описания пользовательских интерфейсов. Помимо полноценных десктопных приложений на платформе Windows, WPF может быть доступным в виде XBAP (XAML Browser Applications), запускаемых в браузере (также на платформе Windows). Помимо этого, Microsoft

объявили о том, что подмножество WPF будет доступно и на других платформах под кодовым названием WPF/E. Летом 2007 года было объявлено официальное название этого продукта: Silverlight.

Silverlight представляет собой кроссбраузерную и кроссплатформенную среду исполнения, базирующуюся на языке разметки XAML. В сентябре 2007 Silverlight 1.0, языком программирования которого является JavaScript, вышел официально. На 2008 год запланирован выход Silverlight 1.1, поддерживающий подмножество CLR, а также DLR (Dynamic Language Runtime), который сейчас доступен в альфа-версии.

Java-апплеты, появившиеся в 1995-м, пожалуй, самое старое из доступных средств создания RIA. Тем не менее, по ряду причин они не получили должного распространения. Слишком большой вес JRE, сложности с определением и установкой нужных версий плагина, долгий старт, традиционный подход к созданию GUI, в котором отсутствовал современный инструментарий для дизайнеров, – все это сделало технологию малопопулярной. К тому же стандартные визуальные темы библиотек пользовательских компонент не соответствовали тенденциям современного графического дизайна. Для решения всех этих проблем фирма Sun предприняла ряд мер. Появился проект Hamburg, позже ставший проектом Consumer JRE, включающий в себя Kernel (базовую JRE, необходимую для старта приложения с малым весом), Deployment Toolkit для определения и установки необходимой версии JRE, Quick Starter (для быстрого старта JRE). Также доступны ранние версии новой векторной кастомизируемой графической темы Nimbus. А главное – JavaFX (именуемая ранее F3) — технология, призванная изменить создание пользовательских интерфейсов на основе нового мощного декларативного динамического языка JavaFX Script.

Таким образом, в докладе сравниваются три основных направления развития RIA от трех производителей: Adobe (Flex/Flash/AIR), Microsoft (Silverlight/WPF) и Sun (JavaFX). За рамками рассмотрения остаются другие, менее популярные технологии (XUL, OpenLaszlo, Curl и т. д.). Также не рассматривается AJAX, который является лишь составным звеном создания RIA в браузере, но не обладает самостоятельной средой исполнения, интегрирующей в себя мультимедиа-возможности и т. д.

В докладе используется множество важных характеристик для сравнения, которые необходимо учитывать при выборе технологии создания RIA, как то: мультимедиа-возможности, поддерживаемые платформы, распространенность, производительность, безопасность, наличие библиотек и компонент, расширяемость, кастомизируемость, интернационализация, deployment, наличие инструментария для разработки, стоимость и многое другое.

## Adobe Integrated Runtime (Adobe AIR): платформа для создания десктопных RIA

*Константин Ковалев*

В ноябре 2003 года фирма Macromedia выпускает первую версию продукта под названием Central, который должен был позволить запускать Flash-приложения посредством традиционного ПО. Из-за ряда причин (необходимость запуска довольно тяжеловесного приложения, из которого запускались остальные; необходимость платить для того, чтобы твое приложение было доступным для установки пользователями, ряд сложностей с отладкой) эта технология не получила распространения, и ее разработка была прекращена. Так или иначе, Macromedia, а после Adobe, извлекли уроки из опыта Central.

Во второй половине 2006 года был анонсирован продукт, с кодовым названием Apollo. С выпуском публичной беты в июне 2007-го появилось официальное название технологии Adobe Integrated Runtime (Adobe AIR).

Adobe AIR призвана объединить на десктопе такие традиционные веб-технологии, как Flash/Flex, HTML/DHTML/JavaScript/AJAX, а также PDF. Таким образом, представляется возможным портировать существующие веб-приложения на традиционную платформу, обогатив их новыми возможностями:

- доступ к файловой системе;
- многооконный интерфейс с возможностью настройки оформления окон;
- любые комбинации Flash, HTML и их взаимодействие;
- drag-n-drop, интегрированный с операционной системой;
- rich clipboard;
- системное меню;
- ассоциации типов файлов с air-приложением;
- поддержка system tray icon;
- встроенная база данных SQLite;
- автоматические обновления.

AIR-приложения доступны на Windows и Mac OS с анонсированной поддержкой Linux в будущем.

Доклад иллюстрирует все основные особенности технологии, способы создания и отладки приложений.

## Работа с данными в формате XML в среде Adobe Flash. Создание standalone приложений с использованием Adobe Flash и Northcode SWF

*Дмитрий Поклонский*

Краткий обзор встроенных классов и методов для работы с XML-данными; XML, XMLNode.

Решаемые задачи: генерация XML, доступ к элементам XML-дерева (задачи получения/отправки данных не рассматриваются).

Недостатки встроенных методов.

Использование библиотеки XPath от xfactorstudio.

Примеры, достоинства, недостатки, область применения.

Создание standalone-приложений с использованием Adobe Flash и Northcode SWF Studio 3: область применения, обзор возможностей API, дополнительные фишки, недостатки.

## Технология Google Web Toolkit

*Аскар Рахимбердиев*

Веб-приложения как полноценная замена традиционного ПО.

### **Платформа**

Историческая перспектива: мейнфреймы, PC, клиент-сервер. Новая модель: SaaS (Software as a Service). Браузер как платформа для разработки приложений. Браузер – виртуальная операционная система. Что мешало раньше? Медленные интернет-соединения и отсутствие технологии. Кандидат на платформу – applets, JavaScript. Applets не сработали. Проблемы JavaScript: несовместимость, сложность разработки. Новые альтернативы: Adobe Flex, GWT (Google Web Toolkit).

### **Технология GWT**

Две задачи: создать интерфейс на стороне клиента и интегрироваться с бекендом.

Интерфейс: компиляция из Java в JavaScript, эмуляция подмножества стандартной runtime-библиотеки Java, библиотека widgets. Для каждого поддерживаемого браузера (IE, FF, Safari, Opera) генерируется свой код. Реализация — использование DOM-модели, вставка фрагментов JavaScript в код Java. Сверху навешивается CSS. Механизм подписки на события (например, ONCLICK). Сделаны обёртки вокруг стандартных

виджетов (div, text, checkbox, button, listbox). Существуют более сложные виджеты, написанные на базе простых (splitter, suggest box).

Коммуникация с бекендом: remote servlets. Сервер должен поддерживать спецификацию servlets, остальное опционально (J2EE, Hibernate). Передача данных – сериализация через RPC, JSON. Сериализуемые типы – примитивные типы, коллекции, maps. Асинхронный интерфейс вызовов – поскольку JavaScript выполняется в браузере однопоточно. Исключения тоже сериализуются.

Цель: responsiveness веб-приложения

- Компилированный код приложения кэшируется навсегда;
- Image bundles – эффективная загрузка картинок;
- Модули: время загрузки приложения vs. время переключения между экранами

Что ещё?

- Интернационализация;
- Поддержка истории браузера для приложения;
- Open source, лицензия Apache 2.0.

### Практика

Что оказалось в реальном проекте. Опасение: слишком тяжёлый клиентский код. Два десятка довольно сложных экранов: 500KB (120KB сжатого) JavaScript кода. Рост объёма кода от функционала нелинейный (медленнее). Ограничения по HTML дизайну (не критичные, но необходимо держать в голове).

Преимущества:

- Приемлемая скорость даже на модемном соединении;
- Привычные технологии;
- Простая интеграция с Java бекендом;
- Desktop-like интерфейс;
- Полноценная отладка;
- Очень легко создавать reusable widgets.

Недостатки:

- Пока мало готовых виджетов;
- Нет поддержки Java 5 (должны добавить в следующей версии);
- Приходится дублировать Entity Beans соответствующими data transfer objects;

- Сложный UI может занимать много памяти и времени CPU на клиенте.

## Анимационные эффекты средствами браузера

*Петр Леонов*

Флеш – это очень круто и это первая технология, о которой вы не узнаете из моего доклада.

Почему? Потому что проприетарно, внутри нет любимых CSS и HTML, неиндексируемо, плохая модель событий, в скриптах слишком долго не было регулярных выражений и замыканий, огромная среда разработки, потребность в компиляции, необходимость создания сайта целиком на флеше.

Почему анимация появилась совсем недавно, да еще в таких количествах;

- ушли от табличной верстки к «дивной»;
- браузеры стали заметно шустрее и аппаратно ускоряемее;
- немаловажно и то, что мы научились правильно использовать JS и CSS.

Как реализована анимация?

Меняем свойство объект по таймеру — вот уже и анимация.

Добавляем тригонометрию и степенные функции и получаем инерцию, скачки и прочую вкуснятину.

Почему не отрендерить в gif или мпег? Вот это вторая и третья технологии, о которых я не расскажу. Но на всякий случай: мпег, сейчас — это флеш, а почему не флеш смотрите выше. Гиф рулит, но то, что он теперь свободен от патента и фотопшоп поддерживает его, нас не спасает. Флеш все равно лучше гифа :)))

Как добиваются плавной анимации (высокого FPS)?

Использовать умение браузера кэшировать часть картинки: с помощью `overflow: hidden`, не использовать прозрачный фон, не перекрывать движущиеся элементы другими, не использовать инпуты и другие gui-элементы.

Очень экономит ресурсы один таймер на все элементы анимации вместо самостоятельных таймеров: разгружает ОС и, главное, перерисовывает страницу один раз для каждого кадра, вместо перерисовки для каждого изменения.

Самое интересное: фонарики — примеры прикольной анимации.

На сладкое – анимация векторных элементов. Специфично для браузеров, но очень приятно.

## Мастер-класс «Геометрия во Flash: кривые Безье второго порядка»

*Иван Дембицкий*

Во Flash для векторного рисования существует только два метода: `lineTo` и `curveTo`, отрисовывающие линию и кривую Безье второго порядка. Контуры любого векторного рисунка во время компиляции также аппроксимируются в отрезки и кривые Безье второго порядка. Но при этом до сих пор нет встроенных методов для получения геометрических свойств этих линий.

В докладе речь пойдет о классах `Bezier` и `Line` проекта [www.bezier.ru](http://www.bezier.ru), созданных для решения этих задач. Поскольку классы не используют объекты Flash и могут быть легко портированы на любой язык программирования, доклад может представлять интерес не только для тех, кто занимается Flash.

## Секция «Веб-стандарты, верстка, HTML 5.0»

### Семантическая вёрстка

*Вадим Макеев*

Ищем смысл: разделение содержимого документа от его представления.

Обзор трёх уровней семантики кода и документа: структура, именование элементов, микроформаты.

Преимущества кода, страниц и сайтов, созданных с учётом семантики.

Начните сегодня: примеры трансформации «супа из тегов» в стройный, понятный и семантически верный код.

### Веб-стандарты в ASP.NET

*Сергей Мезенцев*

Доклад о том, как создавать совершенную клиентскую часть веб-сайта без ограничений в ASP.NET.

Что такое семантическая верстка, веб-стандарты и зачем они нужны в ASP.NET-сайтах.

Модель веб-стандартов в ASP.NET:

- ASP.NET 1.0
- ASP.NET 2.0
- ASP.NET 3.5

Проблемные элементы.

Идеальный сайт в ASP.NET:

- валидный и семантический XHTML;
- повышение общедоступности (accessability);
- эффективный подход к вёрстке сайтов для ASP.NET.

Дополнительные возможности:

- XML/XSLT;
- CSS Friendly Control Adapters.

## Как сделать сайт доступным?

*Юрий Артюх*

Что такое доступность, определение.

Зачем нужна доступность?

- кому нужна «доступность»;
- статистика систем обучения (процент людей с физическими ограничениями);
- видеоролик того, как люди с физическими ограничениями используют IT-продукты.

Как определить, доступен сайт или нет?

- WCAG, 508 и прочие;
- средства тестирования доступности;
- онлайн-чекеры доступности сайта, как с ними работать;
- самый главный критерий: люди.
  - прочитать реальную «плохую» страницу скринридером (JAWS + Windows Eyes) – что бы стало ясно;
  - показать, как воспримет сайт слепой человек.

Как сделать ваш сайт доступным уже сейчас.

Практические методики:

- основы (ссылки, альт, разделение CSS-ХТМЛ, скип ссылки, структура заголовков – best HTML practices);
- как правильно создать сайт для скринридера (с опытом прослушки перед этим);
- как сделать доступными формы, таблицы, JavaScript, AJAX;
- как правильно сделать сайт доступным только клавиатурой;
- доступный Flash;
- доступный медиа-контент (видео, музыка).

Accessibility-линч – анализ доступности известного сайта онлайн.

## Верстка независимыми блоками

*Виталий Харисов*

Что это?

Как это реализуется?

- только class, не id;
- используем префиксы;

- нет классов вне блоков;
- простые и составные;
- полная независимость;
- однотипность верстки похожих блоков;
- модификация постфиксом;
- модификация от контекста.

Префиксы блоков:

- префикс b-;
- префикс g-;
- другие возможные префиксы.

## Тонкий CSS для Internet Explorer

*Павел Корнилов*

Краткое содержание доклада:

- вспомним баги: png, alt&title, label, CSS2 (opacity, hover, first-child, [attribute]);
- основы структур CSS файлов для ie;
- ошибки в классах, там, где их не ждут;
- фильтры, основы;
- файл htc;
- hover.htc и phgfix.htc, учимся искать замену;
- expression, как он работает, и как не работает;
- expression – какие преследуют ошибки. Перекрытие;
- ускоряем expression, делаем незаметно для пользователя;
- position: relative и ссылки. expression для сложных background не замусоривая HTML;
- фишки из CSS3;
- разбор решений.

## HTML 5.0

*Петр Керзум*

Краткое содержание доклада:

- введение: HTML – хаос и порядок;
- краткая история HTML (и факторы его развития);
- текущие рекомендованные стандарты: HTML4, DOM2, XHTML1 (факторы и мотивация);
- конкурирующие факторы дальнейшего развития технологии: WEB 2.0 / семантика документа;
- новейшая история HTML: WHATWG и W3C HTMLWG;

- HTML5: скоро будет =)
  - мотивация: WA1;
  - HTML5 DOM;
  - поддержка семантической разметки документа;
  - HTML5 и XHTML2;
  - конфликты: SGML и FORMS;
  - designer / UA – разделение спецификации;
  - детали: флаги разметки, «score» и модель парсинга;
  - детали: что нового (тэги, концепции, API);
  - детали: чего больше не будет.
- заключение: порядок, хаос и WEB 2.1.

## Мастер-класс «Научим верстать быстро, качественно, дорого...»

*Вадим Макишвили, Павел Корнилов*

Краткое содержание мастер-класса:

- режимы рендеринга;
- правильные и неправильные DOCTYPE;
- способы создания HTML-макета страницы;
- степень поддержки CSS браузерами;
- CSS-фильтры и хаки;
- разные хитрости.

И экстрим! Сможет ли Павел сверстать сайт за 45 минут?

## Секция «Веб-дизайн»

### Нюансы веб-типографики

*Артёмий Ломов*

Понятие экранной типографики применительно к вебу.

Органические отличия экранной типографики от бумажной:

- низкая разрешающая способность;
- неприемлемость традиционных типометрических единиц;
- невозможность жесткого закрепления взаимного расположения слов в пределах колонки набора;
- практическое отсутствие возможности переноса слов.

Правила набора и верстки текста применительно к веб-страницам:

- употребление пробелов;
- употребление неразрывных пробелов и неразрывных диапазонов;
- употребление дефиса, длинного и короткого тире, минуса;
- употребление различных разновидностей кавычек;
- оформление абзацев наборного текста:
  - абзацные отступы и выключка;
  - интерлиньяж;
  - межсловные и межбуквенные интервалы;
- текстовые выделения;
- спорные моменты:
  - неразрывные пробелы после одно- и двухбуквенных предлогов, союзов и частиц в середине предложения;
  - взаимоотношение кавычек и гипертекстовых ссылок.

Технологические нюансы, взгляд с позиций веб-стандартов:

- резкое неприятие распространенной практики использования числовых кодов символов, не предусмотренных рекомендациями W3C (вроде `&#151;`);
- резкое неприятие распространенной практики использования нестандартных элементов HTML (вроде `<nobr>`);
- допустимые единицы измерения для задания кегля шрифта различных версий представления контента (экранной и печатной) при помощи CSS;
- ограничение возможностей задания гарнитуры шрифта при помощи CSS.

# Графический дизайн для веб-среды

*Андрей Окочеников*

Краткое содержание доклада.

1. Основные принципы графического дизайна.
  - 1.1. Гармония. Понятие гармонии. Почему одни сайты выглядят «профессионально», а другие нет?
  - 1.2. Принципы графического дизайна — пропорции, симметрия, контраст, ритм и т.д.
2. Модульный дизайн.
  - 2.1. Понятие модульного дизайна (сетки).
  - 2.2. Историческая справка — золотое сечение, принципы построения, примеры.
  - 2.3. Создание и использование сетки.
  - 2.4. Сетка в вебе. Примеры использования.
  - 2.5. Сетка в пользовательских интерфейсах.
  - 2.6. Инструменты (getLayout.com и другие).
3. Типографика.
  - 3.1. Основные понятия — baseline, kerning, size, ligatures и т.д.
  - 3.2. Типографика и веб. Почему веб плох с точки зрения типографики.
  - 3.3. Основные принципы типографики и их применение в вебе.
  - 3.4. Технологии: CSS, sIFR, и т.д.
  - 3.5. Инструменты: syncotype и др.
4. CSS-фреймворки.
  - 4.1. Зачем нужны CSS-фреймворки и что они делают. Типы фреймворков.
  - 4.2. Обзор существующих решений.
  - 4.3. Преимущества и недостатки фреймворков.
  - 4.4. Разработка собственного фреймворка.

## Управление проектами. Информационный дизайн

*Анна Тихонина*

### **Информационная архитектура**

Понятие информационной архитектуры (информационного дизайна). Командный подход к разработке. Состав команды разработки проекта. Роль информационного архитектора (дизайнера).

### **Построение работы над проектом**

Общие принципы и рекомендации. Семизэтапный процесс ориентированного на пользователя информационного дизайна. Основные роли разработчиков и их распределение.

**Сбор и структурирование информации. Создание концептуальной модели**  
Определение действительных потребностей сайта. Создание и написание пользовательских сценариев. Структурирование информации размещаемой на сайте.

Разработка функционального прототипа. Рисование как способ обмена идеями.

Непрерывный процесс внедрения улучшений, предлагаемых пользователями.

### **Управление наполнением**

Рекомендации заказчику при сдаче «пустого» сайта. Матрица содержания. Наполнение сайта командой разработчиков.

### **Резюме**

Обратная связь с пользователями — выход на новые итерации в совершенствовании проекта. Список материалов по информационной архитектуре. Шпаргалка для руководителя проекта.

## Как выжить дизайнеру в веб 2.0 стартапе

*Алексей Сергеев*

Термины «веб 2.0» и «стартап» стали неотъемлемой составляющей процесса создания новых проектов в Рунете. В докладе рассматриваются особенности веб-стартапов с позиции дизайнера и его места в веб-разработке. Какими качествами должен обладать дизайнер, чтобы быть полноценным членом команды. Какие ограничения необходимо наложить на графический дизайн для быстрого и стремительного развития проекта. Каких ошибок можно избежать и как это сделать. Доклад ориентирован на веб-дизайнеров, заинтересованных в участии в стартапах.

### **Краткий план:**

- что такое веб2.0 и с чем его едят;
- как мы понимаем слово «стартап»; российский стартап, национальные особенности. Зачем создавать свой стартап;
- гибкие методы разработки, что это значит для дизайнера. Эволюционное развитие дизайна проекта vs. революционное — что важно помнить, совершая выбор;
- место дизайнера в стартапе, многоплановость. Качества, которыми должен обладать дизайнер, чтобы быть полноценным членом стартапа;
- ближе к технологиям. Как понять программиста, администратора баз данных и бизнес-аналитика без ущерба для собственного мозга;

- преодолеваем творческий кризис. Какие кризисы бывают, и как самому вытащить себя за волосы из болота;
- «Ф топку» или как принимать сложные решения. «Красота спасет мир», а пока нам надо быстро сделать успешный проект;
- дизайн 2.0. Закругленные углы, градиенты, логотип с зеркальным отражением — что еще? Какие графические особенности сейчас в моде, какие из них необходимо избегать;
- что нужно знать о юзабилити при создании веб-стартапа. Дизайнер в шкуре юзабилити-специалиста;

## Мастер-класс «Интерфейс на экране, на бумаге и в жизни»

*Артем Горбунов*

Артем Горбунов рассказывает о том, как дизайн пользовательского интерфейса и технологии информационного дизайна решают задачи бизнеса и повседневной жизни.

## Секция «Дизайн интерфейсов, юзабилити, accessibility»

### Качество человеко-компьютерного взаимодействия: подход в международных стандартах

*Константин Самойлов*

Под интерфейсом сегодня понимается не только то, что пользователь видит на экране, а все взаимодействие с системой, которое он осуществляет. Поэтому качество интерфейса включает в себя как технические, так и человеческие факторы и рассматривается как обобщающий показатель качества всей системы. Этот принцип заложен в стандарт ISO 9241 — «Руководство по юзабилити» и используется в новой серии стандартов по качеству программных продуктов ISO 25000.

В докладе будут рассмотрены практические преимущества и ограничения этого подхода, его направление развития, использование и перспективы в России.

### Состояние потока (FLOW) как цель проектирования взаимодействия

*Денис Бесков-Доронин*

#### Что такое «состояние потока»

Состояние потока — психическое состояние, в котором человек полностью включен в то, чем он занимается, что характеризуется деятельным сосредоточением, полным вовлечением и успехом в процессе деятельности.

Корни и взаимосвязи: буддизм, боевые искусства, спорт.

#### Основные свойства:

- Ясные цели (различимые ожидания и правила).
- Концентрация и фокус внимания — высокая степень концентрации на ограниченной сфере внимания (человек, занимающийся деятельностью, имеет возможность на ней концентрироваться и глубоко в нее погружаться).
- Потеря чувства самоосознания — слияние действия и осознанности.
- Искаженное восприятие времени.
- Прямая и незамедлительная обратная связь (успехи и неудачи в процессе деятельности очевидны, так что поведение может быть изменено по мере необходимости).
- Равновесие между уровнем способностей субъекта и сложности задания (деятельность не оказывается для субъекта слишком легкой или сложной).

- Ощущение полного контроля над ситуацией или деятельностью.
- Деятельность сама по себе воспринимается как награда, так что она осуществляется без усилий.

### **Взаимосвязь потока и продуктивности**

С точки зрения деятельности, помимо положительных эмоций, сопровождающих состояние, оно также сопровождается высокой эффективностью из-за бессознательности, автоматизма действий.

### **Пирамида пользовательских качеств**

На чем основывается достижение состояния потока? На выполнении следующих условий (снизу вверх):

1. преисполненность смыслом;
2. приятность;
3. привычность, очевидность;
4. удобство;
5. надежность;
6. безопасность;
7. функциональность.

### **Фокусы традиционных классов систем**

Нечеткий класс системы	Приоритетные аспекты
Промо-сайт	Эстетичность Вовлекаемость
Игра	Вовлекаемость
Финансовое ПО	Безопасность
Одноразовое ПО	Функциональность
Компоненты повторного использования	Расширяемость
Военные, медицинские системы	Надежность Безопасность
Публичные веб-системы	Производительность Удобство
Системы автоматизации	Функциональная полнота Надежность

Системы автоматизации бизнеса, хотя и имеют свои резервы в этом смысле, пока не нуждаются в новых качествах, т.к. компаний, которые организовывали свой бизнес согласно принципу достижения счастья и удовольствия от работы, пока очень мало.

Высокая конкуренция в среде онлайн-веб-систем и коробочных продуктов заставляет бороться за всё новые уровни качеств. Таким образом,

FLOW-ability представляется вполне интересным и стоящим рубежом, который им предстоит взять. Но как?

### **Игры как естественная среда для достижения FLOW**

Вместе с разработкой коробочных продуктов автоматизации персональной деятельности и веб-систем существует параллельный мир игр, в котором свои законы организации, свои принципы и методы ведения разработки. И в играх, как можно понять из определения «потока», достигается это состояние, иначе бы они не были играми!

#### **Основные приёмы игровой среды:**

1. коллекционирование, накопление —> привязанность;
2. очки, баллы —> рейтинги, статистика;
3. уровни —> новые возможности;
4. обратная связь —> мастерство, погружённость, удовольствие (ключевой приём для потока);
5. обмен;
6. персонализация.

Демонстрация применения игровых приемов в веб-системах

Перспективы использования игровых техник в прочих классах систем для достижения состояния потока

## **Расширенное понимание доступности, ее основные составляющие и методы обеспечения**

*Денис Бесков-Доронин*

### **Введение**

В последние годы обрел определенную популярность термин «доступность», под которым, однако, разными кругами специалистов понимаются различные узкие свойства систем. Число пользователей ИКТ-систем растет, все более важным становится умение создавать такие интерфейсы, которые бы позволяли эффективно решать задачи удовлетворять потребности пользователей. Автор предлагает расширить возможное понимание доступности с точки зрения интересов пользователя, а также рассмотреть методы ее обеспечения.

### **Цель работы**

Показать комплексность понятия «доступность» с точки зрения интересов пользователя ИКТ-систем, выведя его за рамки специализированных юридических, эргономических и технологических трактовок.

Описать методы обеспечения составляющих доступности.

## **Тезисы**

Применительно к системам, построенным на активном использовании информационно-коммуникационных технологий (ИКТ), можно говорить о доступности:

1. информации;
2. услуг (сервисов).

В данном докладе речь пойдет преимущественно о первой категории, как более фундаментальной. С другой стороны, предоставление информации можно считать разновидностью услуги.

## **Частные случаи толкования термина «доступность»:**

1. юристы — фактическое наличие информации в системе;
2. эргономисты — возможность воспользоваться системой, воспринять информацию для людей с ограничениями по здоровью;
3. ИТ-технологи — способность системы предоставлять информацию на различных устройствах доступа без существенных потерь и искажений.

## **Типовой сценарий доступа и использования информации**

Рассмотрим типовой сценарий получения и использования информации при помощи ИКТ-систем:

1. сформулировать проблему, для решения которой может быть полезна искомая информация;
2. обратиться к нужной системе — поставщику информации;
3. найти и ознакомиться с найденной информацией в системе;
4. уточнить свое понимание проблемы с учетом полученной информации;
5. получить помощь в решении проблемы, используя те или иные методы навигации по системе;
6. использовать приобретенные знания для разрешения проблемы.

## **Типовые проблемы при доступе и использовании информации**

С каждым из этих шагов на практике может быть связан ряд проблем:

1. пользователь затрудняется сформулировать свою потребность;
2. невозможность получить доступ к информации:
  - а. пользователь не знает названия системы, содержащей информацию и способа получения доступа к ней;
  - б. классификаторы, каталоги и справочники не дают понятных указаний относительно того, в каких системах искать нужную информацию;
  - с. глобальные поисковые системы часто работают неэффективно;
3. информация в системе неполна и неактуальна;

- a. система не содержит нужной информации;
  - b. система содержит неактуальную, устаревшую информацию;
4. неудобная форма представления информации:
- a. система имеет затрудненную подсистему навигации;
  - b. информация представлена в закрытых, проприетарных форматах;
  - c. подача информации не адаптирована и ее восприятие для рядового пользователя затруднено.

Особенно актуальны эти проблемы в области публичного веба и государственных сайтов.

### **Составляющие доступности**

Понимание того, как протекает сценарий доступа и использования информации, а также с какими проблемами сталкивается пользователь в его ходе, дает основания для выделения существенных составляющих доступности и, таким образом, переопределения этого понятия в расширенное:

1. фактическое наличие и актуальность информации;
2. возможность найти эту информацию;
3. подготовленность информации для восприятия.

### **Методы обеспечения составляющих доступности и участники**

**Фактическое наличие и актуальность:**

1. определение потребностей — маркетологи, аналитики;
2. подготовка и поставка информации — служба обеспечения информацией, служба сбора и поставки данных, редакция, владельцы информации.

**Возможность найти информацию:**

1. реклама, продвижение — рекламные службы, службы веб-оптимизации;
2. техническая доступность — ИТ-технологи;
3. создание эффективной навигации — глобальные поисковые службы, проектировщики взаимодействия;

**Подготовленность для восприятия:**

1. обеспечение технической переносимости — ИТ-технологи;
2. адаптация, толкование текста — копирайтеры, журналисты, редакторы, специалисты предметной области;
3. обогащение подачи — информационные дизайнеры;
4. удобство восприятия — эргономисты.

## Заключение

Доступность информации — комплексная характеристика системы, ее обеспечение требует согласованной работы специалистов из различных отраслевых категорий. Провал в одной из составляющих доступности может означать потерю полезности для пользователя.

## Usability-проектирование. Dark Side of the Source

*Артем Мошков, Олег Безуглов*

### Срез

- веб-студии;
- интернет-компанияи.

Те, кто сегодня создает:

- корпоративные сайты;
- каталоги и магазины;
- новостные сайты.

Сюда относятся и поставщики коробочных решений.

### Ситуация на рынке

Usability, несмотря на широкую известность, до сих пор не стала регулярной дисциплиной при разработке интернет-систем. Основное приложение — публичная часть сайтов, нацеленная на потребителей. Часто без внимания остаются:

- административные разделы;
- служебные экраны;
- внешние программные интерфейсы (API).

... и растут, как сорняки.

### Место встречи 1

Типичные функции административных разделов:

- управление содержанием;
- управление структурой;
- управление дизайном;
- мониторинг/SEO;
- взаимодействие с аудиторией, управление.

Это едва ли не все места, в которых ЦА и есть заказчик, точнее его работник. Место пересечения заказчика, как ЦА (со всем его спектром требований), и исполнителя. Место встречи изменить нельзя.

## **Место встречи 2**

Иногда на исполнителя нападают другие исполнители, но их взаимоотношения, равно как искусство проектирования API, мы обсуждать не будем. Да, вы знаете, программисты тоже потребляют ваш продукт. Пусть это даже опосредованное потребление при помощи поисковых роботов и прочих агрегаторов.

Могу заметить, что при нынешних тенденциях развития сети, качественное API продукта вскоре будет конкурировать с его клиентской частью. Особенное внимание стоит уделять коробочным продуктам, которые вы уже поставляете или только собираетесь.

## **Место встречи 3**

Служебные экраны:

- вход/регистрация;
- восстановление пароля;
- корзина;
- оплата;
- поиск и т.д.

По важности приравниваются к административным разделам. Это еще не административная, но уже и не публичная часть сайта.

Исполнение этих экранов критично настолько же, насколько вы оцениваете важность своей услуги.

Насколько часто служебные тексты пишут программисты и насколько часто это смешно?

### **Так в чем же проблема?**

Заказчик выбирает продукт с его interaction одновременно с выбором исполнителя. Т.е. принимает и покупает usability в составе продукта.

Нужно ориентироваться на рынке, но что происходит после приобретения?

Получая экраны «as is», заказчик предпочитает потратить деньги не на их адаптацию, а на учебный курс.

Неудобный или нетривиальный интерфейс — отличный аргумент для ежемесячного обслуживания. А мы и рады. Такое паразитное существование процветает и тормозит прогресс обеих сторон.

### **...Usability формируется исполнителями!**

Вы можете ответить себе, кто на самом деле заказывает продукт?

Пестовать usability продукта могут графические дизайнеры, программисты, менеджеры всех мастей, сам заказчик (ориентируясь на свой опыт или просто эмоции)...

Кто пользуется продуктом?

Пользуются им как раз не исполнители, а контент-менеджеры, работники склада, служба поддержки и т.п.

Насколько бы радостным и плодотворным не было сотрудничество первых, без прямого привлечения пользователей и аналитиков, качественным такой продукт будет лишь на бумаге.

#### **Решения поставки интерфейсов:**

- реализация на базе шаблонного решения;
- на базе коробочного решения;
- на базе чужого решения;
- индивидуальная разработка;
- экспертное улучшение проблемных мест.

Как правило, универсальные и коробочные решения ориентированы на конвейер и содержат универсальный interaction. Это выгодно для вас. Редко — для клиента.

Клиенты приводят клиентов. Выводы?

Все обобщенные разработки — это сочетание best practices, в которые мы свято верим. :-)

Настолько ли общим является бизнес ваших клиентов?

#### **Контролируемые мутации**

При адаптации/изменении подсистемы, внесении «несущественных» изменений, интерфейс проявляет свою инертность. Это общая проблема пользователей и исполнителя.

Растет энтропия системы вообще, и интерфейсов в частности.

Индивидуальные проекты — самый полный контроль.

Если вы не эксперт — привлекайте. Вы отвечаете за продукт. Если вы хотите создать вокруг себя плотную сеть клиентов, вы должны отвечать.

## **Организация процесса/сопровождение продукта**

Часто проектная группа успевает «все сделать» только в публичной или административной части. Почему?

Одни из самых ощутимых причин потери целостности продукта:

- дисциплина разработчиков;
- гегемония заказчика;
- дефицит ресурсов (время, деньги, нервы).

Объясните заказчику, как лучше решать его задачу. Вы должны это сделать, если вы цените этого клиента и дорожите будущими.

Сохраняйте и закрепляйте целостность своих продуктов.

## **Аналитика**

Откуда брать данные для анализа?

Вовлекайте в процесс закрепления interaction/usability не заказчика с деньгами, а будущих пользователей. Они пригодятся для «разведки боем», выявления существующих проблем и формирования карты потребностей.

Учитывайте расширение продукта, увеличение объемов данных (степени свободы бизнес-модели). В пределах естества вашего продукта, разумеется.

## **Прототипирование**

Элементарное прототипирование на бумаге и карточки. Это экономит ресурсы.

Создавайте и тестируйте кейсы «по контракту» и их отклонения (personas).

Контролируйте и протоколируйте все метаморфозы: требований, программных изменений, адаптации к операционной среде; накапливайте статистику.

Не отдавайте процесс на откуп замкнутой группе разработчиков. Везение не бывает систематичным.

## **На ком тестировать?**

Начните с себя.

Совсем не фееричный, но эффективный комплекс простых и дешевых методов (больше одного):

- замер времени работы/числа ошибок;
- успешность обучения;
- отзывы/констатация результатов (осторожно, пожелания! будьте объективны).

Практикуйте тестирование коробочных решений. Чем более дистанцирован/ распространен продукт, тем дотошнее должно быть тестирование.

## Адаптация продуктов Microsoft для людей с ограниченными возможностями (accessibility)

*Петр Диденко*

### **Почему accessibility — очень важный вопрос для Microsoft**

Во всем мире живет огромное количество людей с ограниченными возможностями — инвалидов, людей с ослабленным зрением и другими физиологическими недостатками. Microsoft очень серьезно относится к возможностям своих продуктов, которые позволяют таким людям полноценно использовать наше ПО.

### **Какие продукты содержат accessibility features**

Большинство популярных настольных продуктов изначально проектируются так, чтобы ими могли пользоваться инвалиды. Какие средства конкретно и в каких именно продуктах предлагает Microsoft сегодня:

- продукты семейства Windows;
- Microsoft Office;
- Internet Explorer 7;
- средства разработки семейства Microsoft Visual Studio;
- SQL Server;
- устройства — клавиатуры и мыши.

### **Исторический экскурс**

Когда и какие конкретно accessibility-возможности предложил пользователям своего ПО Microsoft.

### **Как инвалиды используют accessibility-возможности**

Подробнее о том, как инвалиды пользуются accessibility в продуктах Microsoft на примере конкретных людей и организаций.

## **Будущее accessibility**

Планы Microsoft по развитию accessibility-возможностей. Новые технологии Microsoft для дизайнеров и usability-профессионалов.

## **Особенности проектирования интерфейсов для начинающих пользователей**

*Андрей Золотов*

Любой продукт не может быть использован эффективно и продуктивно неким абстрактным «пользователем», а рассчитан на вполне определенных пользователей. При проектировании интерфейса необходимо четко представлять себе тех, кто будет с ним работать, а также учитывать их способности и навыки. Именно это позволит создать по-настоящему удобный и успешный продукт.

Сегодня лишь для небольшого количества людей информационные системы (ИС) являются объектом работы, большинство же используют их как вспомогательный инструмент. Адвокаты, биржевые брокеры, кассиры и бухгалтеры вынуждены осваивать вновь появляющиеся ИС в перерывах между своей основной работой. В такой ситуации становится актуальной специализация разрабатываемых интерфейсов под начинающих пользователей.

В своем докладе я хотел бы сначала охарактеризовать начинающих пользователей, рассказать про их особенности такие, как:

- уже существующий опыт работы с ИС;
- привычки;
- повышенное внимание к деталям;
- неуверенность в себе.

Затем дать конкретные рекомендации по разработке интерфейсов для новичков. Затронуть следующие составляющие:

- терминология и тексты;
- используемые элементы управления;
- используемые метафоры;
- контекстные подсказки.

Дать рекомендации по форме и содержанию сопроводительного справочного материала.

Ключевые слова: юзабилити; проектирование интерфейсов; начинающие пользователи.

## Мастер-классы «Юзабилити-тестирование? Легко! (или как в домашних условиях протестировать программный продукт)»

*Катерина Умнова*

«Разработчики и пользователи по-разному смотрят на программные продукты». Это утверждение стало почти прописной истиной, в результате чего пользователям часто бывает сложно работать с компьютером. Однако, гораздо больше половины программных продуктов все еще создаются без привлечения специалистов по разработке интерфейсов. Тем не менее, определить основные болевые точки разрабатываемой или уже готовой системы можно самостоятельно: для этого нужно провести небольшое юзабилити-тестирование.

В докладе я расскажу о том, когда можно и нужно проводить юзабилити-тестирование, и как это можно сделать в «домашних» условиях. То есть, как «на коленке»:

- определить ключевые задачи и сформировать задания;
- подобрать подходящих пользователей;
- провести тестирование;
- разобрать и оценить результаты.

Кроме этого, мы с вами рассмотрим примеры того, как этот процесс:

- может получиться на практике;
- от чего может не получиться;
- и на что обратить внимание, чтобы повысить вероятность успеха.

## Мастер-класс «Юзабилити аспекты проектирования пользовательских интерфейсов в среде Microsoft Expression Blend»

*Сергей Швецов*

Новое средство для создания пользовательских интерфейсов от Microsoft. Что это? Очередной «монстр» от известной компании или полезная технология, которая наконец-то примирит два лагеря врагов-соперников: программистов и юзабилитистов?

В своем докладе я хочу рассказать о новой технологии Windows Presentation Foundation (WPF) от Microsoft и языке описания пользовательских интерфейсов XAML, подробно осветив следующие темы:

- построение «богатых» пользовательских графических интерфейсов при помощи XAML. Функциональность. Взаимодействие XAML и процедурного кода.
- Microsoft Expression Blend современный инструмент создания пользовательских интерфейсов. Почему Blend? Обзор и краткая характеристика приложений работающих с XAML. Специфические инструменты дизайнера интерфейсов. «Богатая» графика. Особенности построения интерфейсных элементов в Blend.
- Плюсы и минусы Expression Blend. Взгляд в будущее проектирования интерфейсов. Стоит ли овчинка выделки?

Ключевые слова: WPF, XAML, Microsoft Expression Blend, юзабилити, проектирование интерфейсов.

## Секция «Шаблонизаторы»

### Почему XSLT?

*Сергей Бережной*

Краткое содержание доклада:

1. Лирическое вступление. Цель доклада.
2. Физическое вступление:
  - а. что такое «шаблонный движок»;
  - б. проблематика: разделение труда в большом динамично развивающемся проекте.
3. Ближе к телу. Философия XSLT (versus, «классический шаблонизатор»):
  - а. «Суперфичи»:
    - нативная рекурсия по входным данным;
    - Xpath;
    - паттерн матчинг.
  - б. «Не супер, но фичи»:
    - наследование шаблонов;
    - переопределение переменных.
4. Что дальше? Приемы, усиливающие эффективность «фич».

### Эффективное использование XSLT

*Александр Мартынов и Александр Ермолаев*

Краткое содержание доклада:

1. кратко о технологии XML+XSLT. Когда оправдано применение;
2. почему мы используем libxslt (конечно-же из-за того, что он самый быстрый):
  - а. какие есть альтернативы;
  - б. сравнение производительности;
  - с. сравнение возможностей.
3. Эксперименты с производительностью:
  - а. разумное использование ключей;
  - б. именованные шаблоны;
  - с. минифицирование самих xslt-документов;
  - д. проблемы с функциями exslt.org, какие быстрые, какие не очень;
  - е. функции, которые мы часто используем — переписывание их с xslt на C;
  - ф. влияние кодировки документа на производительность;
  - г. динамическое создание элементов против явного их указания;

- h. подключаемые файлы — компромисс между удобством разработки и производительностью;
  - i. разумное использование переменных;
  - j. аккуратное применение рекурсии (существует опасность превышения глубины вложения);
  - k. профилирование.
4. Типичная структура проекта на XML+XSLT:
- a. каждой странице свой XSL плюс общая библиотека;
  - b. импорты против инклюдов. Плюсы и минусы;
  - c. много маленьких шаблонов или один большой? Что хорошо, что плохо;
  - d. опасная XSLT функция document() (неуправляемые таймауты — большая проблема);
  - e. статика и псевдостатика (данные, которые обновляются не очень часто, полезно хранить в виде статического XML).
5. Рекомендации программистам:
- a. удобный XML. Что это такое?
    - Комфортные форматы даты;
    - минусы большой вложенности;
    - соглашения по именованию тегов (подчеркивание, минус, точка и т.п. в именах);
    - одинаковые имена на разных уровнях вложенности;
    - XML-помойка.
  - b. Что лучше: тег или атрибут?
  - c. Выводить ли в дерево пустой тег?
6. Библиотека стандартных компонентов:
- a. пейджер (листалка);
  - b. форма логина;
  - c. копирайт;
  - d. docbook2HTML;
  - e. xforms.

## Как проще и эффективнее строить работу с унаследованным от других разработчиков xsl-кодом

*Евгения Фирсова*

Краткое содержание доклада:

1. Первый этап — понимание. Приемы, помогающие быстрее подружиться с legacy-кодом.
  - a. Написание документации помогает понять, что и как работает.
  - b. Статистическое исследование кода: иногда нам важно знать, какие блоки кода, в принципе, существуют, выделить повторя-

- ющие элементы, оценить частоту появления тех или иных конструкций.
- с. Взлом «черных ящиков» — что делать, если код написан слишком запутанно или финальный результат трансформации зависит от исходного xml-документа неизвестного формата.
2. Второй этап — изменения/рефакторинг.
    - а. Как выбрать, с каких частей проекта стоит начинать рефакторинг.
    - б. Как вносить изменения безопасно.
    - с. Почему нельзя называть точные сроки.
    - д. Разбиение кода на отдельные модули.
    - е. Отделение ресурсов от кода.
    - ф. Оптимизация xsl-кода.

## Мастер-класс «Применение XSLT на стороне клиента»

*Алексей Остапенко*

Краткое содержание мастер-класса:

1. когда и как применять;
2. использование: все «за» и «против»;
3. организация работы;
4. ограничения браузеров в поддержке языка;
5. XSLT и JavaScript:
  - а. возможности;
  - б. реализация;
  - с. сложности/ограничения;
  - д. примеры (в сравнении с другими решениями);
  - е. XML -> JSON;
  - ф. сортировки;
  - г. pure XPath.

## Секция «Мобильные технологии (WAP, PDA-проекты)»

### Юзабилити программ для мобильных устройств

*Федор Ежов*

Мобильные устройства завоевывают мир, и очень многие пользователи оценили для себя удобство и те широкие возможности, которые предоставляют смартфоны и карманные компьютеры в повседневной жизни и бизнесе. Поэтому разработчикам программного обеспечения особенно важно разрабатывать не только функциональные, но и удобные продукты, которые будут понятны как продвинутым пользователям, так и обычным домохозяйкам. В данном докладе рассматриваются устройства на базе Windows Mobile, но общие рекомендации могут быть применимы и к устройствам на базе других систем.

Краткая ретроспектива развития интерфейсов и возможностей:

- HPC, Palm-Size PC, Auto PC;
- WinCE 3.0 (Cedar) (кнопка «Start», «плоский» дизайн окошек, полноэкранные диалоги, отсутствие кнопки закрытия окна);
- WM 2003 Second Edition (поддержка VGA и режимов Square\ Landscape);
- Windows Mobile 5(появились soft-keys!);

В настоящее время наблюдается тенденция отказа от стилуса как от средства управления и идет переход на one-hand navigation(QWERTY-смартфоны и keypad) и touch-navigation (устройства с которыми можно работать посредством пальца, подобные Apple iPhone и HTC Touch).

Заниматься юзабилити продукта для мобильного устройства нужно начинать на самых первых этапах разработки (параллельно с написанием проектных спецификаций). Общие фазы работы юзабилитиста могут быть следующими.

- Разработка прототипов параллельно с разработкой базовых спецификаций. Это может быть как работающий прототип, так и максимально детализированные скриншоты будущего продукта.
- Отслеживание в процессе работы соответствия разрабатываемых интерфейсов и стандартных рекомендаций производителей.
- Проведение face-to-face юзабилити-тестирования с целевыми группами пользователей.
- Самостоятельное использование продукта с ранних альфа-версий (если это возможно).

- Активное участие в бета-тестировании, сбор пожеланий и уточнение неясных моментов.
- Совместная работа с технической поддержкой после выпуска продукта на рынок или его внедрения для определения оставшихся «узких» мест.
- Постоянное использование мобильного устройства, чтобы понимать, что на нем удобно, а что нет.

Для юзабилита основные отличия продукта для мобильных устройств от продуктов на десктопе следующие:

- очень маленькая рабочая область (экран);
- отсутствие мыши и клавиатуры, других устройств управления и навигации;
- слабый процессор и ограниченная память;
- меньшая цветовая гамма.

Основные ошибки, которые допускаются при разработке мобильных приложений:

- one-hand navigation — это то, что должно быть обязательно;
- продукт должен обеспечивать работу в landscape, square, portrait режимах;
- отсутствие поддержки VGA и WXGA;
- нестандартные и несвойственные элементы управления (например, использование комбобоксов в WM Standard);
- нетипичная для мобильного устройства навигация;
- использование жестко прошитых шрифтов и цветовой гаммы;
- большое время запуска приложения;
- слишком большое время реакции на действия пользователя;
- сложные меню и сокрытие в них функциональности;
- невозможность настройки размера шрифтов, используемых в программе, в том числе clear-case;
- только для Windows Mobile Professional: нужно не забывать про SIP!

## Мобильные магазины и платежи

*Александр Штучкин*

Вследствие возрастающей популярности коммуникаторов и других мобильных устройств, прослеживается всё большая заинтересованность больших игроков в создании платформ для мобильных магазинов. Данная область считается очень перспективной, особенно в свете объемов продаж контента для мобильных телефонов, используя платные СМС-сообщения.

Уже сейчас существует множество вариантов создания магазина для мобильных устройств, однако, глядя на их интерфейсы, можно смело утверждать, что данная область находится в ранней стадии развития. Только недавно компания Microsoft в последней версии Windows Mobile 6 выпустила платформу для создания мобильных магазинов «Microsoft Marketplace». Не отстает и Nokia со своим приложением «Catalogs», а также другие компании, такие, как Sony и Handango.

В докладе выражается попытка выработать необходимые рекомендации по проектированию мобильных магазинов. Процесс покупки на мобильном устройстве можно условно разделить на 2 этапа:

- просмотр каталога товаров, сравнение и выбор покупаемого товара;
- оплата товара и скачивание;

В обоих случаях проектировщики интерфейсов сталкиваются с юзабилити-проблемами, характерными как для мобильных устройств, так и для онлайн-магазинов. В докладе рассматриваются следующие проблемы и методы их решения:

- эффективное использование малого экранного пространства на коммуникаторах;
- навигация по каталогу без использования стилуса;
- описание продуктов — на какой уровень пользователя рассчитывать;
- уменьшение задержек обновления интерфейса при медленных подключениях к интернет;
- выбор способа платежа: СМС, кредитные карты и другие способы;
- необходима ли корзина товаров? Упрощение оплаты товара для пользователя;
- автоматическая регистрация купленных программ.

## Секция «Качество, тестирование»

### Автоматизация тестирования веб-интерфейсов с использованием Selenium

*Виталий Помазенков*

Краткое содержание доклада.

1. Кратко про автоматизацию тестирования веб-интерфейсов.
  - a. зачем оно вообще нужно;
  - b. возможности автоматического тестирования;
  - c. типы ошибок;
2. Что такое Selenium:
  - a. область применения;
  - b. основные возможности;
  - c. архитектура.
3. Что такое наша (наша = компании Яндекс) обертка AQuA вокруг Selenium:
  - a. основные возможности;
  - b. архитектура;
  - c. отчеты и автоматическая генерация сценария;
  - d. подробности реализации;
  - e. как устроена среда тестирования.
4. Как писать автоматические тесты:
  - a. простейший тест;
  - b. полезные и вредные советы.
5. Проблемы автотестирования:
  - a. решенные и которые еще предстоит решить.

### Автоматизация тестирования сложных Flash-интерфейсов

*Александр Комлев*

Автоматическое тестирование функционально сложных flash-элементов на веб-страницах относится к тому кругу задач, о котором принято говорить: «Хорошо бы сделать это, но не сейчас». Но когда руки, наконец, добираются до одной из таких задач, оказывается, что готовых и удобных решений не существует, есть только несколько идей, которые вроде бы и подходят, но не совсем и нужно большое время на разработку решения почти с нуля. И опять задача откладывается до лучших времен. В очередной раз о ней вспоминают в момент, когда назревает серьезная необходимость в этом; приходится думать о том, как быстро и эффективно реализовать задумки.

Попытки автоматизировать рассматриваемый процесс предпринимались не один раз и каждый — с переменным успехом. В итоге, ни о какой универсальности, простоте и кросс-платформенности не могло быть и речи. В чем же недостатки предыдущих решений? Что мы хотим получить от нового решения? Какими средствами мы собираемся добиваться необходимого результата? Всему этому и посвящен доклад.

План доклада.

1. Введение:
  - a. постановка задачи;
  - b. существующие решения и почему это «не работает» или «работает неправильно»;
  - c. критерии поиска «правильного» решения.
2. Концепция решения:
  - a. компонент ExternalInterface class + JavaScript;
  - b. интеграция с используемыми средствами;
  - c. примеры реализации.
3. Разработка автоматических тестов:
  - a. написание, запуск тестов, анализ результатов;
  - b. проблемы и недостатки выбранного решения.
4. Итоги:
  - a. где и как можно эффективно использовать такой подход;
  - b. потенциальные проблемы при решении задач.

Что такое XSS и как их искать

*Алексей Капранов*

## Автоматизированное тестирование смешанных web- и win-интерфейсов

*Екатерина Ивахина*

Данный доклад посвящен применению средств автоматизации в задачах тестирования смешанных web- и win-интерфейсов. Тема рассматривается на примере панели инструментов Rambler-Ассистент.

План доклада:

1. Описание проекта Rambler-Ассистент.
  - а. Область применения.
  - б. Использование.
  - с. Установка/обновление/удаление.
2. Цели и объекты автоматизации.
  - а. Ускорение процесса проверки.
  - б. Регрессионное тестирование.
3. Выбор средства (QTP, SilkTest, TestComplete, Robot).
  - а. Сравнение плюсов и минусов.
  - б. Возможность применения в рамках нашей задачи.
4. Разработка скриптов.
  - а. Возможности.
  - б. Проблемы и методы их решения.
  - с. Улучшения.
5. Итоги.
  - а. Сравнение показателей ручного и автоматизированного тестирования.
  - б. Достоинства и недостатки выбранного решения.

## **ДЛЯ ПОМЕТОК**

---

## **ДЛЯ ПОМЕТОК**